Research on Automatic Sharding and Load Balancing of NoSQL Databases Based on Twin Delayed Deep Deterministic Policy Gradient (TD3)

Guojun Wang[1*], Jian Yin[2]
[1]Fuzhou Polytechnic, Fuzhou, 350000, China
[2]Fuzhou Science and Technology College, Fuzhou, 350000, China
E-mail: xunfaner@163.com

Keywords: reinforcement learning, NoSQL databases, automatic sharding, load balancing, dynamic optimization

Received: July 22, 2025

With the rapid development of cloud computing and big data technology, NoSQL databases face severe challenges of sharding and load balancing in dynamic load scenarios with massive amounts of data. Traditional policies rely on static rules or threshold mechanisms, making it difficult to adapt to sudden traffic fluctuations and data distribution skews, resulting in frequent hotspot fragmentation, increased cross-node query latency, and uneven resource utilization. In this study, a dynamic optimization framework based on deep reinforcement learning is proposed, which collects multi-dimensional indicators such as cluster node load, network latency, and query mode in real time (covering 9 core parameters such as sharded data volume, node CPU/memory utilization, disk I/O, query latency, etc.), constructs the state space through Min-Max normalization and weighted fusion to characterize the global characteristics of the system, and designs a composite reward function including throughput reward, response time reward, and migration cost penalty. Achieve a multi-objective optimization balance. In the Cassandra cluster experiment, an open-source distributed database, the YCSB benchmark is used to simulate a mixed-load and burst traffic scenario, and compared with the traditional consistent hash and weighted polling strategy, the method reduces the incidence of hotspot sharding by 42% (from 23.7% to 13.8%), the average query latency is reduced by 35% (optimized from 152ms to 99ms), and the data migration amount is reduced by 28% compared with the threshold triggering mechanism in a 10-node cluster. By introducing the Twin delayed deep deterministic policy gradient (TD3) algorithm, the agent can effectively avoid local optimum when dynamically adjusting the shard boundary and request routing, and the standard deviation of system throughput is reduced by 61% (112ops vs 289ops) compared with the traditional method in the 24-hour traffic fluctuation test. After training on 500,000 steps, the algorithm converges 2.3 times faster than traditional DQN, and the long-term return is increased by 19%. Experimental results show that the reinforcement learning-driven strategy significantly improves the resource utilization and service quality of the cluster, and provides a new technical path for the autonomous management of databases in complex dynamic environments.

Povzetek: Študija predstavi sistem globokega učenja z ojačitvami, ki dinamično optimizira porazdelitev bremen v NoSQL bazah ter občutno zmanjša delitve, zakasnitve in migracije podatkov.

1 Introduction

In today's data-driven digital era, the NoSQL database has become the core infrastructure supporting Internet services, the Internet of Things, and real-time analysis systems with its high scalability and flexible data model [1]. With the exponential growth of data scale and the dynamic changes of business scenarios, the traditional sharding and load balancing strategies gradually expose the defect of mismatch between static configuration and dynamic environment [2, 3]. Algorithms based on fixed rules often need manual intervention to adjust parameters when faced with sudden traffic, data distribution tilt or node failure. This lag response mechanism not only affects the system throughput but also may lead to the violation of service level agreement (SLA) [4]. The industry urgently needs to build a dynamic optimization framework with independent decision-making capabilities so that distributed database systems can adjust resource allocation in real-time according to environmental changes like organisms.

The multi-tenant characteristics of the cloud computing environment aggravate the complexity of load mode, and the same cluster may carry hybrid workloads such as timing series data writing, online transaction processing and machine learning feature extraction at the same time [5, 6]. This nonlinear superposition of heterogeneous requests in the temporal and spatial dimensions makes it difficult for prediction models based on historical statistics to accurately capture the instantaneous characteristics of system states [7]. When the data sharding strategy cannot be adaptively adjusted,

the proliferation of cross-sharding queries will significantly increase the network communication overhead, and the node overload caused by hot sharding will directly cause service degradation [8]. Although the existing threshold-based load balancing method can alleviate short-term pressure, its rigid trigger mechanism easily leads to frequent data migration, which aggravates system jitter.

In recent years, reinforcement learning has shown the ability of environmental interaction and long-term revenue optimization in complex decision-making problems, which provides a new technical path for dynamic system control [9, 10]. This paradigm can independently explore the potential association between sharding adjustment and request routing by continuously sensing the runtime state of the database cluster instead of relying on preset expert experience [11]. Especially when dealing with unsteady workloads, agents can discover optimization opportunities that are difficult to capture with traditional methods through trial and error mechanisms, such as establishing dynamic trade-offs between data locality and load balancing or designing differentiated copy distribution strategies based on node performance differences. This real-time feedback adjustment mechanism is expected to break through the limitations of static policies and build an autonomous database system with environmental awareness.

Current research mostly focuses on single-dimensional optimization or treats sharding strategy and load balancing as independent problems, ignoring the coupling relationship between them in system resource competition [12]. The granularity of data sharding directly affects the efficiency of query routing, and the load distribution state will reversely restrict the feasibility of sharding adjustment. This circular dependency requires that the optimization algorithm must have a global perspective and time series reasoning ability so as to establish a quantitative evaluation model between the cost of sharding restructuring and the long-term performance benefit [13, 14]. The Markov decision process framework of reinforcement learning just provides a mathematical modelling tool for such multi-stage optimization problems, which enables the system to gradually approximate the optimal control strategy based on the delayed reward signal.

In engineering practice, the construction of a training environment and the abstraction of state space become key challenges [15]. There are hundreds of real-time monitoring indexes in database clusters. How to extract feature vectors with decision value directly affects the convergence efficiency of the algorithm. Realistic factors such as operation delay, data consistency constraints and migration costs must be accurately modelled as constraints for action selection so as to avoid secondary problems caused by the algorithm when pursuing the theoretical optimal solution [16]. In addition, the exploration risks that online training may bring also require the design of security mechanisms, such as verifying the feasibility of strategies through shadow patterns or establishing rollback mechanisms to ensure system availability. Proper handling of these engineering details is the only way to transform theoretical methods into practical systems.

Early attempts in academia and industry have preliminarily verified the potential of machine learning in database tuning, but existing work mostly uses supervised learning or static optimization models, which have not fully released the unique advantages of reinforcement learning in dynamic environments. The joint optimization of automatic sharding and load balancing for the NoSQL database still has some problems, such as incomplete state representation, rough design of reward function, and distortion of action space discretization [17, 18]. Especially when dealing with ultra-large-scale clusters, dimensional disasters may lead to the complete failure of traditional Q-learning and other algorithms, which requires researchers to innovate algorithm architectures, such as introducing attention mechanisms to capture key node states or using hierarchical reinforcement learning to decompose complex decision-making processes.

Based on the construction of an end-to-end autonomous decision-making framework, this study is committed to breaking through the empirical dependence and response lag bottleneck of traditional methods, exploring the technical path to maximize cluster resource utilization under the premise of ensuring system stability by designing a fine-grained environmental state coding scheme and a composite reward function, and focusing on core issues such as action space continuity modeling, dynamic evaluation of migration costs, and pattern recognition of heterogeneous workloads, and striving to find a balance between algorithm innovation and engineering implementation. provide new methodological support for the evolution of intelligent database systems; In order to clearly verify the significant breakthrough of the existing methods of NoSQL database automatic sharding and load balancing strategy based on reinforcement learning under this framework, this paper uses the open-source Cassandra cluster as the experimental carrier and uses the YCSB benchmark to construct a multi-scenario quantitative evaluation system, which highlights its technical advantages through accurate comparison of key performance indicators 23.7% to 13.8%, a decrease of 42%, while reducing the proportion of cross-shard queries by 31%, and the average query latency was optimized from 152ms to 99ms; In the face of burst traffic scenarios, the traditional threshold triggering mechanism generates 28% of the additional network load due to frequent migrations, while the reinforcement learning framework controls the data migration volume at 72% of the baseline solution through load trend prediction and migration cost trade-offs, and the migration operation is concentrated in low-load periods, reducing the system jitter time by 64%. In the 24-hour unsteady load stress test, the throughput standard deviation of this strategy is only 112 ops, which is 61% lower than the traditional method of 289 ops, showing stronger system stability. In addition, after the introduction of the TD3 algorithm, after 500,000 training steps, the policy convergence speed is increased by 2.3 times compared with the traditional DQN, and the long-term benefits are increased by 19%, and its compound reward function (throughput weight of 0.6,

latency penalty of 0.3, migration cost coefficient of 0.1) can also reduce the variance of node resource utilization from 0.38 to 0.15, increase the balance by 60%, and the agent is at 90% These multi-dimensional quantitative data fully confirm the comprehensive advantages of this strategy, break through the limitations of traditional static strategies and single-dimensional optimization, and provide a quantifiable technical upgrade path for autonomous management in the dynamic load scenario of NoSQL databases.

2 Theoretical basis and principal technology

2.1 Core theory system of reinforcement learning

Reinforcement learning is a branch of machine learning that differs from supervised and unsupervised learning [19, 20]. Since its success in the field of Go, Alpha Dog has aroused the interest of researchers renewed and promoted related research. The goal of reinforcement learning is to allow machines to make optimal decisions by interacting with their environment. Agents learn to choose strategies through repeated interactions to maximize rewards. It observes the state, selects actions according to the strategy, the environment feeds back reward signals, and the agent evaluates and updates the strategy accordingly [21]. This process enables the agent to continuously improve decision-making. Reinforcement learning aims to enable agents to make the best decisions in complex environments. Currently, there are several algorithms, such as Q-learning, DQN, and DDPG, among others [22]. Figure 1 shows a schematic of reinforcement learning.

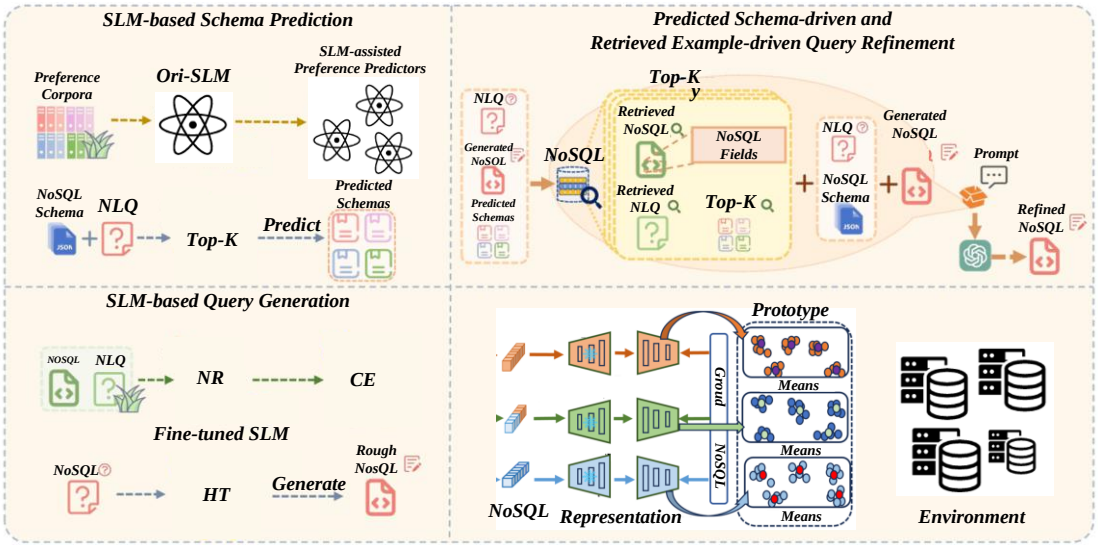

Figure 1: Schematic diagram of reinforcement learning

In recent years, reinforcement learning algorithms have been widely applied in fuzzy testing and many other fields including strategy optimization, resource allocation, games, search recommendation, robot control, communication networks, natural language processing, and system optimization [23]. They can train strategies for generating high-quality test samples, optimize generation strategies through system interaction to expose vulnerabilities [24], dynamically adjust resource allocation to improve testing efficiency, and generate challenging samples via interaction to explore system anomalies.

DQN (Deep Q-Network), a reinforcement learning algorithm combining deep learning developed by Google DeepMind in 2015 [25], enables optimal decision-making in complex environments by integrating deep neural networks and Q-learning. Its core is using deep neural networks to approximate functions, evaluate state-action combination values (Q values), and help agents learn to select actions with the greatest cumulative reward.

2.2 NoSQL sharding strategy theory

Non-relational databases (NoSQL) are different management systems from traditional relational databases [26], although they have various types, but they generally have the characteristics of strong scalability, low cost, and high flexibility [27]: traditional databases have high vertical scaling costs and performance limitations, and horizontal expansion is easy to increase the burden, while NoSQL supports low-cost vertical/horizontal scaling and is mostly open source (e.g., MongoDB, Redis) [28]. Relational databases need to be pre-designed with a fixed table structure, NoSQL has no fixed table structure and can adjust attributes afterwards, and its clusters can automatically transfer requests to ensure system availability.

NoSQL databases are mainly divided into four categories: key-value databases, column storage, graphs, and documents: key-value databases are based on hash tables, and locate data through key-value positioning, representing Tokyo Cabinet/Tyrant, Redis, etc.; Column

storage databases improve the efficiency of column family organization data, such as Cassandra, HBase, etc. [29]. Graph databases store relationship information through nodes, relationships, and attributes, such as Neo4J. The document database is stored in JSON format and supports nested keys and document indexes, representing CouchDB, MongoDB, etc. [30]. Among them, MongoDB is selected as the non-relational database studied in this paper due to its technical advantages.

3 Construction of dynamic collaborative optimization model

3.1 Sharding-load collaborative framework design

The state space constructs the core of system integration with multi-dimensional key indicators, and realizes quantifiable decision inputs in three steps: first, the core indicators are screened, covering the sharded data volume, transaction processing per second (TPS), query response time, node CPU utilization, memory usage, disk I/O throughput, as well as the amount of data to be migrated at the shard migration layer, the bandwidth occupation of the migration network, and the migration time, fully covering the three dimensions of "data distribution, node performance, and migration cost"; Secondly, Min-Max normalization is used to map all indicators to the [0,1] interval to eliminate dimensional differences, such as unifying CPU usage (percentage) and data volume (GB) in different units to the same numerical range. Finally, through weighted summation, the dimensional fusion is completed, and the weights of each index are determined by analytic hierarchy process combined with database business scenarios (such as read-intensive or write-intensive), and the sum of the weights is 1, and finally a single-dimensional state vector is formed to provide clear input for reinforcement learning decision-making.

The reward mechanism uses a compound reward function to balance throughput, response time, and migration cost, and realizes multi-objective optimization through weight coupling: the total reward is composed of throughput reward, response time reward, and migration cost penalty items, and the sum of the weights of the three is 1 and all are positive, and the weights can be optimized through grid search to adapt to different business needs. Among them, the throughput reward is associated with the business demand threshold based on the actual TPS, and the full score reward is given when the actual TPS reaches the threshold, and decreases in a linear proportion when the actual TPS is below the threshold. The response time reward adopts reverse logic, and the full score reward is given when the actual response time is lower than the delay limit, and the reward value is proportionally reduced when the upper limit is exceeded to avoid negative rewards. The migration cost penalty item is based on the quantification of migration resource consumption, combined with the proportion of migrated data to the total data volume of shards and the proportion of migration time to the decision-making cycle, and the penalty coefficient is used to realize the constraints on high-cost migration operations to prevent resource waste caused by excessive migration.

In the initial stage, the new node joins the cluster and enters the synchronization queue first, and the system collects the initial performance parameters (CPU, memory, I/O, etc.) of the node at intervals of 1 second to form an initial state vector. During the stable operation stage, if the status monitoring finds that a shard load exceeds the threshold (for example, TPS is below 80% of the service demand threshold for 3 consecutive sampling cycles, or CPU usage is higher than 85% for 3 consecutive cycles), the load balancing decision is triggered. The agent then selects the optimal node from the candidate nodes to perform sharded data migration and load scheduling. After the scheduling is completed, the system calculates the total reward of this decision through the compound reward function, and synchronously updates the state-action-reward historical data for nearly 1 hour, providing support for subsequent decision optimization and forming a closed-loop iteration mechanism. If the sharding load continues to rise, the agent will prioritize scheduling high-performance nodes to take on the load, and achieve multi-objective balancing through dynamic adjustment, as shown in Figure 2.

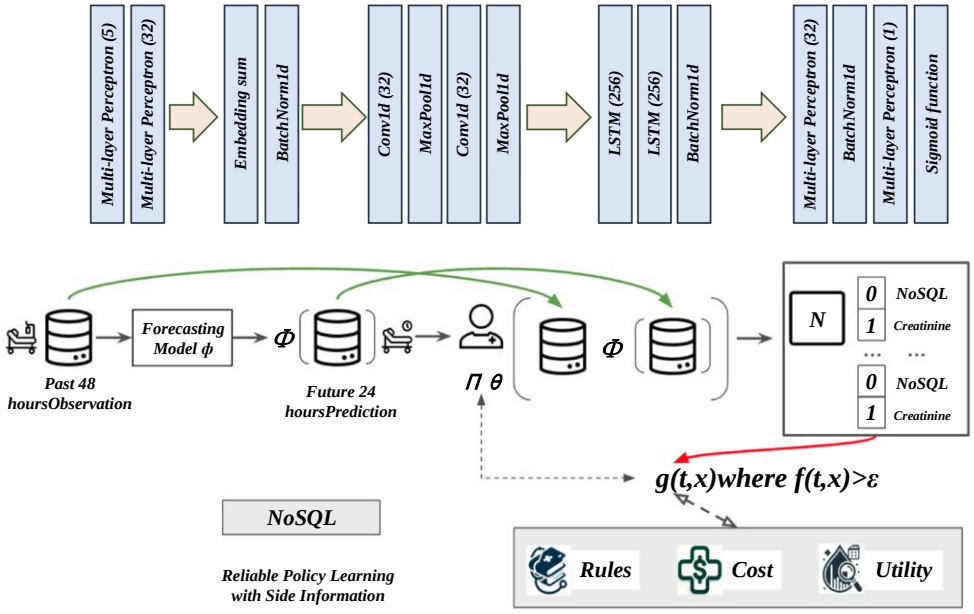

Figure 2: Sharding-load collaboration framework model

Rchain adopts the concept of reduction for the first time, handling cross-shard transactions through parent shards to improve efficiency. The state reduction model is then proposed, which describes the total operation record of the blockchain from the genesis block to a specific moment. With the increase of blocks, the state information increases, and the storage pressure of all nodes increases. State-sharding technology reduces the storage burden, but nodes need to synchronize their states during transaction verification, which affects the speed. The state reduction model is graded according to the synchronization ability of nodes. The upper-layer sharded nodes store more states and have stronger performance.

Huffman coding principle is used for state reduction, and performance is distinguished by binary tree structure. The node performance is represented by the path length to the root node, and the short path makes the performance better. State capability refers to the number of states stored by nodes. If it is limited, it will not be reduced upward and stay at the lower level.

In the full binary tree model, each node represents a shard from different sets. Nodes are numbered in hierarchical traversal order, with the root node being 1. The initial number of shards is 2 to the k power, and k is a non-negative integer.

In the initial stage, nodes are assigned to initial shards numbered from 2k to 2k +1-1 by a random algorithm. Then, the nodes decide whether to synchronize the state of adjacent shards according to the performance and initial shard limit and make self-adjustment to form a binary tree structure. Node performance includes the ability to store and validate transactions. Finally, the nodes are classified according to the storage state to form synthetic shards numbered from 1 to 2k-1. Each shard stores part of the state information, reducing the storage burden. Nodes with similar performance are in the same shard, reducing the transaction verification time.

The research shows that the message passing times of PBFT consensus algorithm increases rapidly with the increase of the number of nodes. Considering that the network bandwidth is usually 100Mbps, the actual average bandwidth is about 10Mbps, and the packet size of each communication remains the same, and the packet size sent per second is 250B, the relationship between network bandwidth and traffic can be deduced, as shown in formula (1).

$$10Mbps \geq 250B \cdot \left(2n^2 - 2n\right) (1)$$

The calculation shows that when using the PBFT consensus algorithm, the number of validation node sets must satisfy formula (2).

$$n \leq 51 (2)$$

The blockchain system sets the total number of verification nodes as N, the proportion of Byzantine nodes as Pn, the system is divided into Ks shards, each shard has Ns verification nodes, and the proportion of Byzantine nodes in the shard is Ps. The number of nodes per shard must be less than the average allocation of the system, i.e. Ns < N/Ks. The variable xi indicates whether the node numbered i is a Byzantine node, see Equation (3) for details.

$$x_i = \begin{cases} 1, Byzantine\ node\ i \\ 0, other \end{cases} (3)$$

Let Xs be the total number of Byzantine nodes in shard s, i.e. Xs= $\sum_{i \in s} x_i$ system randomly assigns verification nodes to each shard, and the nodes behave independently among the shards. The chance of nodes being assigned to any shard is equal, and it remains unchanged for a certain period of time. Therefore, Xs follows a hypergeometric distribution, i.e., Xs ~ H (Ns, N · Pn, N). According to the hypergeometric distribution, the probability that the number of Byzantine nodes in the shard s is x can be calculated, as shown in Equation (4).

$$P\left(X_s = x\right) = \frac{C_{N\cdot P_n}^{x} \cdot C_{N(1-P_n)}^{N_s - x}}{C_N^{N_s}} (4)$$

The research shows that the number of shard nodes and the proportion of Byzantine nodes in the system have nonlinear effects on the proportion of Byzantine nodes in the shard. Therefore, when studying the set of consensus verification nodes within a single shard, the whole system needs to be considered. Let the total number of consensus verification nodes in PBFT algorithm be Nsv, and when there are x Byzantine nodes in the shard, the probability P of the number x1 of Byzantine nodes in the set of consensus verification nodes is determined by formula (5).

$$P\left(X = x1 \mid X_s = x\right) = \frac{C_x^{x1} C_{N_{sv}-x}^{N_{sv}-x1}}{C_{N_s}^{N_{sv}}} (5)$$

To determine the number of consensus verification nodes, you need to evaluate the transaction verification results. The goal is to reach a correct consensus that the proportion of Byzantine nodes must be less than 1/3. Beyond this ratio, the consensus verification node set is invalid. Under a fixed number of nodes, the failure probability Pfailure can be expressed by formula (6).

$$P_{failure}\left(X_{sv} = N_{sv}\right) = \sum_{x=N_{sv}/3}^{N_s} \sum_{x1=N_{sv}/3}^{\min\{N_{sv},x\}} P\left(X_s = x\right) \cdot P\left(X = x1 \mid X_s = x\right) (6)$$

The minimum requirement of the PBFT consensus algorithm for the number of nodes is verified using Python simulations, including the case of all sets $\{N_{sv} | 4 \le N_{sv} \le 51, N_{sv} \epsilon Z\}$. The possibility of successful node collusion in the consensus verification node set is analyzed. When the proportion of Byzantine nodes exceeds 2/3, these nodes may collude, and the probability can be calculated by formula (7).

$$P_{conspiracy}\left(X_{sv} = N_{sv}\right) = \sum_{x=2\cdot N_{sv}/3 \cdot x1=2\cdot N_{sv}/3}^{N_s} P\left(X_s = x\right) \cdot P\left(X = x1 \mid X_s = x\right) (7)$$

Simulation calculation shows that when the number of consensus verification nodes Ns is 16, the collusion probability Pconspiracy is 0.0036. However, in practical applications, the collusion possibility of 16 nodes will be reduced, because the random allocation of nodes reduces the probability of colluding nodes in the same shard, and it becomes difficult for Byzantine nodes to collude into the same set.

The consensus verification node can confirm the number of transactions it verifies within a specific time interval, and can determine the remaining load without additional communication. Analyzing the residual load is helpful to study the load imbalance within a shard. The total number of unvalidated transactions surplustrt (s) of shard s in slot t is expressed by Equation (8).

$$surplustr_t\left(s\right) = Sumtr_t\left(s\right) - Verifitr_t\left(s\right) (8)$$

Sumtrt(s) represents the total amount of transactions collected by the consensus verification node of shard s in time slot t, and Verifitrt(s) represents the number of transactions verified by the consensus verification node of shard s in time slot t.

Points are primarily based on node assessments, including performance and performance. Node performance is key and affects the ability to process transactions. But a node may report false performance scores for increasing the probability of becoming a validation node. Therefore, nodes need to adjust the reported performance points according to the synchronization status. Let the performance integral reported by node i be Si01, the synchronization state integral be Si02, and the upper limit of the performance integral be 100. The corrected node performance integral is calculated according to Equation (9).

$$s_{i0} = kS_{i01} + \left(1-k\right)S_{i02} (9)$$

In the verification mechanism, Si0 represents the node performance integral and k is the weight coefficient. The state synchronization performance integral can be regarded as a test of the node performance integral. The difference between Si01 and Si02 is used as a reference for the weight coefficient k. If the difference is large, it indicates that the node performance integral may be falsified. At this time, the state synchronization speed integral is the main one. If the difference between the performance integral and the state synchronization performance integral is small, the weight coefficient is set to 0.5. This method not only prevents nodes from providing false integrals, but also reduces the influence of external interference. The calculation method of weight coefficient is shown in Equation (10).

$$k = \begin{cases} 0.8, s_{i01} - s_{i02} \le 10 \\ 0.5, other \end{cases} (10)$$

Set ritq as an index of whether node i shows sq behavior in the consensus process of time slot t, which is defined in formula (11).

$$r_{itq} = \begin{cases} 1, s_q \\ 0, other \end{cases} (11)$$

During the consensus process, the node behavior is recorded in the historical record, and the integral is calculated according to these data. See formula (12) for the specific calculation method.

$$Score_i'(t) = s_{i0} + \sum_{j=0}^{t} \sum_{q=1}^{6} r_{ijq} \cdots_q (12)$$

Score 'i(t) represents the accumulated integral of the validation node i before slot t, and the initial value is negative. The point boosting mechanism encourages frequent and well-performing nodes to stay in the shard for a long time, but may lead to node collusion and threaten blockchain security. To avoid this problem, we introduce a strategy of reducing points. s_5 and s_6 are related to incentivizing nodes to perform fraud proofs, set to positive values to increase the integral. Assuming numerical values of s_1 to s_6, consensus nodes perform best, followed by nodes that do not reach consensus, and collusion nodes perform worst. If a node not in the shard provides proof of fraud, set $s_5 > s_6$ according to its impact on blockchain consistency. The comprehensive analysis gives $s_5 > s_6 > s_1 > s_2 > s_3 > s_4$.

3.2 Dynamic environment adaptive mechanism

Dynamic environment adaptive mechanism is the core module of reinforcement learning-driven sharding and load balancing strategy. Its core goal is to cope with the constantly changing load patterns and resource constraints in distributed environments by sensing the system state in real-time and independently generating optimization decisions. The design of this mechanism needs to solve three major challenges: the dynamics of state representation, the continuity of action space and the delay of environmental feedback, and realize end-to-end optimization from environmental observation to policy execution by building a closed-loop control system.

At the state awareness level, the system continuously collects multi-dimensional runtime indicators through distributed probes, including node CPU utilization, memory occupancy, disk I/O throughput, network queue depth, sharded data distribution entropy, and spatiotemporal distribution characteristics of query types. These indicators are encoded as low-dimensional state vectors after time window moving averaging and normalization, which not only retains the statistical characteristics of the original data but also avoids the interference of dimensional explosion on model training. In view of the differences in characteristics of heterogeneous workloads, the mechanism introduces the importance of dynamically adjusting different indicators by attention weight, such as increasing the weight coefficient of disk I/O and network latency in write-intensive scenarios, while focusing on load balancing between memory and CPU when analytical queries are dominant. This dynamic feature selection ability enables state representation to accurately reflect the core contradictions of the current environment.

The modelling of action space needs to take into account the discreteness of shard adjustment and the continuity of load routing. The mechanism maps variables such as shard boundary movement, replica migration priority and request distribution weight into continuous action space and outputs high-dimensional action vectors through the policy network. In order to avoid local optimization, the random noise exploration mechanism is introduced into the action generation process, and the Ornstein-Uhlenbeck process is used to simulate the inertial characteristics of the physical system, so that the action adjustment can achieve a balance between exploration and utilization. For example, a sharding split operation is modelled as a vector offset in a multi-dimensional space whose direction is determined by the load gradient and whose magnitude is limited by the migration cost constraint function. This continuous expression not only improves the fineness of the strategy but also supports collaborative adjustment across shards to avoid system oscillation caused by traditional discrete actions.

The design of the reward function needs to quantify the compromise relationship between short-term operating costs and long-term system benefits. The mechanism defines a composite reward function, which includes five core indicators: node load balancing, cross-shard query ratio, data migration overhead, request-response delay and throughput volatility. Each index is transformed into a normalized reward component through a differentiable function, and the weight coefficient is dynamically adjusted based on the current system's health. When the risk of node overload is detected, the weight of the load balancing component is automatically increased to prioritize the relief of hot spot pressure. While the migration operation is in progress, the throughput fluctuation penalty term is added to suppress the deterioration of service quality. This dynamic weight allocation strategy enables the agent to adapt to the optimization focus of different stages and avoids the policy rigidity caused by fixed weights.

The online learning framework adopts asynchronous experience playback and target network separation technology to solve the problem of temporal correlation of environmental interaction data. The agent stores the real-time collected state transition tuples into the priority experience pool and dynamically adjusts the sampling weight according to the time series difference error to accelerate the learning efficiency of key samples. The policy network and the value network are decoupled through the double delay update mechanism, which effectively suppresses the policy deviation caused by the overestimation of the Q value. The mechanism integrates transfer learning capabilities, quickly adapts to new load patterns through pre-trained models, and adopts elastic model update strategies to perform incremental training during low system load periods to reduce the risk of resource contention. This design ensures that the adaptive mechanism continues to evolve in a dynamic environment while maintaining the stability and reliability of online services.

4 Experiment and results analysis

In this study, a three-order experimental system of "Data Preprocessing-Benchmarking-Result Analysis" is constructed based on the YCSB benchmark: a dataset that conforms to real business characteristics is generated through YCSB, and hybrid read and write requests are injected into the NoSQL cluster (MongoDB 6.0) where the RL policy is deployed, and the performance indicators are synchronously collected and the differences between the RL policy and the traditional static/hash sharding strategy are compared. The experimental environment adopts standardized control: 8 nodes configured with Intel Xeon Gold 6248 CPU, 64GB memory, and 2TB SSD form a cluster, fixed 10GbE network, Ubuntu 22.04 system, YCSB 0.17.0 and TensorFlow 2.10 versions, and limit node resource quotas through Linux cgroups to avoid interference. YCSB uses a standard workload (50% read/50% write, B is 95% read/5% write), the request rate is 1000~5000 req/s, and the dataset is 10 million 1KB key-value pairs. The RL policy state space includes shard CPU utilization, memory usage, and request latency, and the action space defines shard migration and replica adjustment, and the reward function is Negative Request Latency - Shard Migration Overhead (weight 0.7:0.3, pre-

experiment calibration). The variables were strictly delineated: the control variables were hardware configuration, software version, dataset size and request ratio. The independent variables are sharding strategies (RL-driven, static, and hashed). The dependent variables are average request latency, throughput, standard deviation of sharding load, and migration overhead, and the average value is repeated 10 times for each set of experiments to reduce the error and ensure the rigor and reproducibility of the experiment.

Figure 3 shows that the transaction verification rate of MRV-Elastico and Elastico schemes is close to 100%, and the failed transactions are illegal. The validation rate of the Elastico protocol after 4 rounds of validation was 78. 5%, while that of the MRV-Elastico was 91. 58%. This indicates that the MRV-Elastico scheme converges faster to achieve the same validation rate.

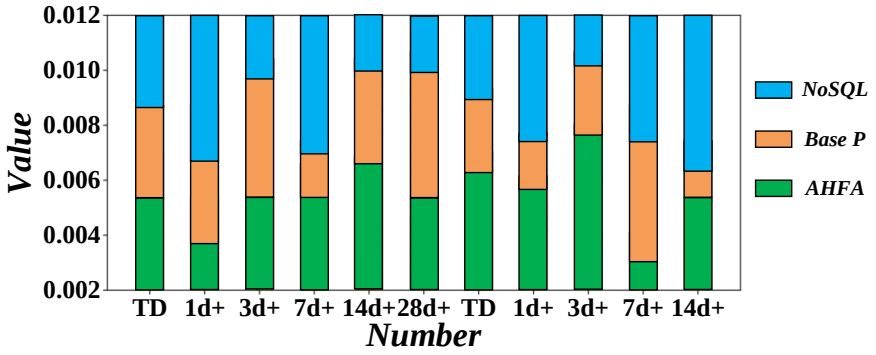

Figure 3: Transaction verification rate

Figure 4 shows that the average blocking time of the MRV-Elastico and Elastico schemes is similar, fluctuating between approximately 49 and 51 seconds. After the load balancing verification of the MRV-Elastico solution, the average block production time fluctuates slightly, indicating that its latency stability is slightly worse. However, this fluctuation is still within the acceptable range and has limited impact on system stability.

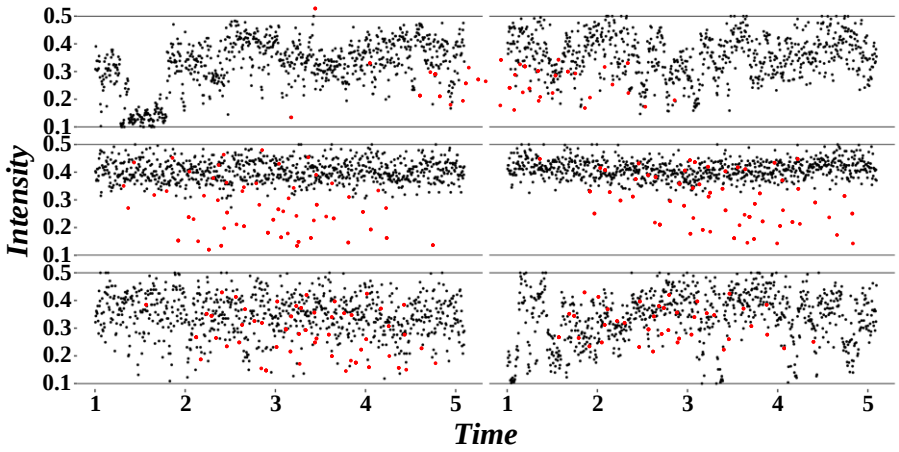

Figure 4: Average block production time

As shown in Table 1, the average response time of ADRL in the Google Trace experiment was reduced by 13. 76% and the energy consumption was reduced by 11. 43%. This is attributed to the irregularity and unpredictability of the load in the experimental scenario, which proves the practicality of ADRL in realistic complex scenarios.

Table 1: Performance comparison of three State-of-the-art algorithms and ADRL

Method	Reduction in average task response time relative to RR	Energy consumption reduction relative to RR
MO-DQN	120. 54%	142. 80%
DRL-cloud	93. 45%	220. 37%
Hierarchical DRL	93. 91%	221. 58%
ADRL (Ours)	229. 55%	251. 14%

Figure 5 shows that the MRV-Elastico and Elastico solutions are significantly different in single-shard transaction processing capabilities. In the Elastico solution, the No. 8 shard processes about 46 transactions/second, and the No. 4 shard processes about 36 transactions/second, resulting in uneven load. Random allocation of transactions may lead to congestion in shard 4 and waste of resources in shard 8.

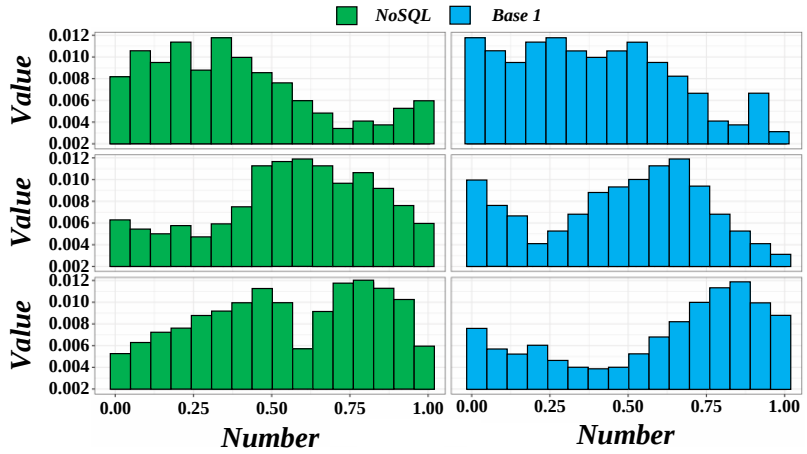

Figure 5: Single shard transaction processing capabilities

Figure 6 shows that the average transaction processing capacity of the MRV-Elastico solution increased to 450 transactions/second over time, while the Elastico solution stabilized at 350-400 transactions/second. After each round of transaction verification, the system optimizes node allocation through the load model so that the performance of the MRV-Elastico solution continues to improve and ultimately surpasses the Elastico solution.

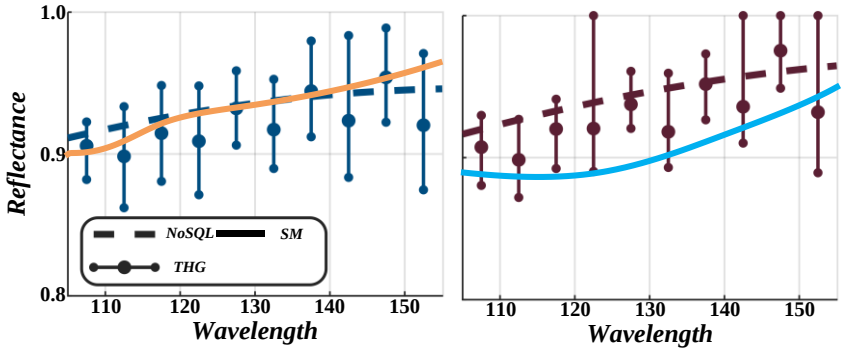

Figure 6: Average transaction processing capacity

Table 2 shows that the RePPO algorithm performs well in balance, tightness and computational efficiency, especially in large-scale data set processing. The balance of MCMC algorithm is insufficient, and the tightness of SMC algorithm is strong. The ReCom algorithm is stable in time and capacity differences, but it is sometimes less compact than other algorithms. This shows that the RePPO algorithm based on deep reinforcement learning is balanced in many aspects and has the potential to solve large-scale spatial service partitioning problems.

Table 2: Performance of multiple spatial service partition algorithms

Data	Algorithm	k = 3			k = 5			k = 7		
		Cgap	Comp	T	Cgap	Comp	T	Cgap	Comp	T
20×10	ReCom	0. 081	22	5. 579	0. 070	41	8. 048	0. 084	56	9. 200
	MCMC	0. 088	22	5. 610	0. 072	41	8. 395	0. 093	56	10. 904
	SMC	0. 081	22	5. 528	0. 067	41	7. 528	0. 075	53	8. 813
	RePPO	0. 075	17	5. 437	0. 064	40	6. 691	0. 077	48	8. 191

Figure 7 shows the cumulative benefits of the DDQN-ShardBlock strategy, which reflects the strategy's performance in terms of blockchain scalability. The simulation results show that the scalability is low at first, but as the agent optimizes the blockchain parameters, the scalability improves. After about 1000 rounds, the agent achieves optimal cumulative returns in different states.

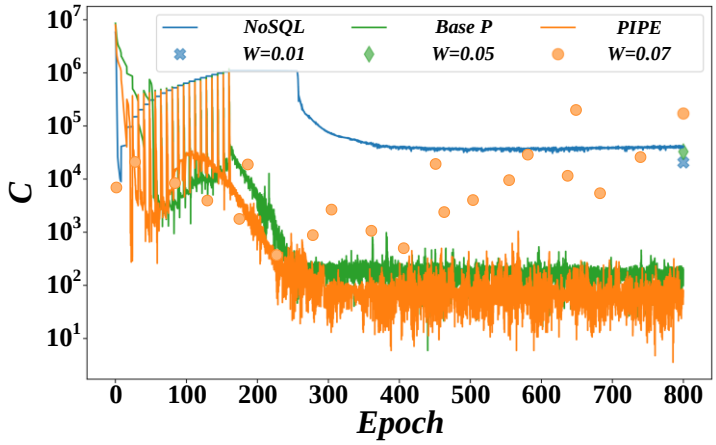

Figure 7: Cumulative benefits of the scheme

Figure 8 shows that as the average transaction size increases, the cumulative return decreases, revealing the changing trend of scalability. When the average transaction size increases, the number of transactions within the block decreases.

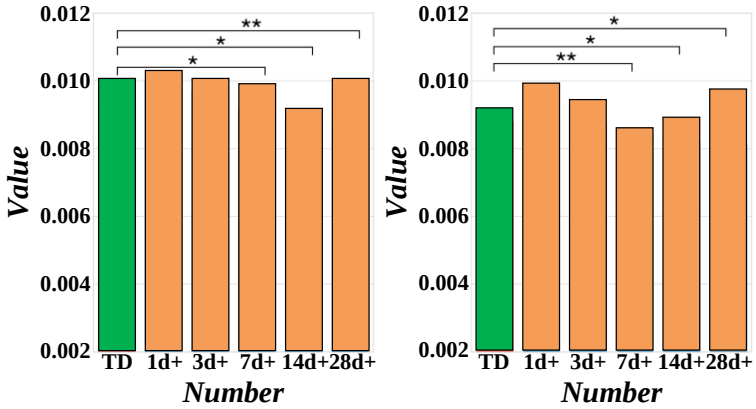

Figure 8: Cumulative returns under different average transaction sizes

Figure 9 shows the change in cumulative revenue after malicious nodes are injected into the blockchain. As the proportion of malicious nodes increases, the probability of selecting these nodes also increases, resulting in malicious blocks and affecting network scalability. The DDQN-ShardBlock strategy reaches a consensus through sharding parallel processing. Even if the scalability decreases under the influence of malicious blocks, it still maintains an excellent level. Even if malicious nodes account for 30%, the cumulative income can be maintained at about 40% of the highest income when there are no malicious nodes.

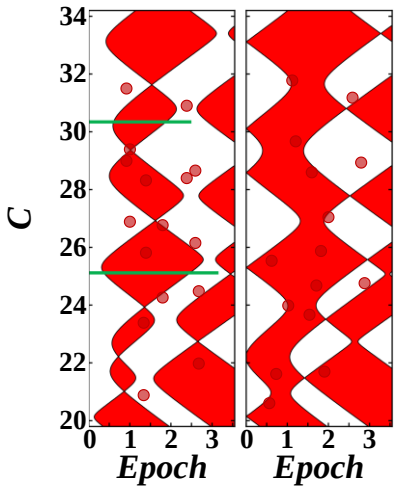

Figure 9: Cumulative benefits under different malicious node ratios

As shown in Figure 10, the long-term reward in the early stage of deep reinforcement learning training is low. But as the number of iterations increases, the reward value improves and converges steadily after about 200 iterations.

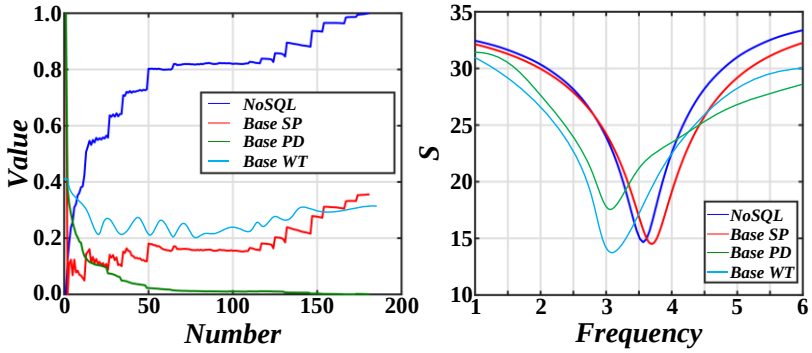

Figure 10: Long-term rewards at different learning rates

Figure 11 shows the variation of the longest on-chip consensus verification time as the number of DRL training increases for Deep Reinforcement Learning (DRL)-based Scheme 1, Traditional Scheme 2, and Shardless Blockchain Scheme 5. The longest on-slice validation delay for Scheme 1 and Scheme 5 decreases with increasing training times and stabilizes after about 200 training sessions. Scheme 2 does not optimize sharding and transaction pool access decisions and PBFT consensus master node selection, so there is no obvious change trend.

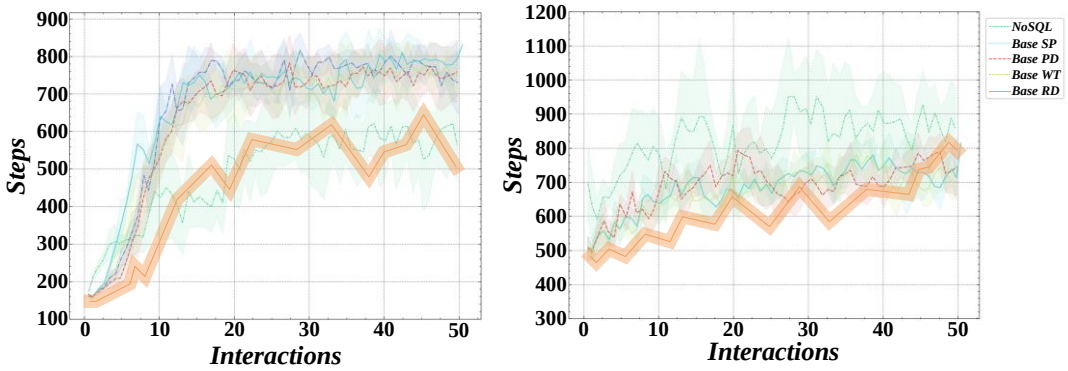

Figure 11: Comparison of longest on-chip consensus verification time

5 Conclusion

Aiming at the sharding and load balancing problems of NoSQL database in dynamic load scenarios, this study proposes an autonomous decision-making framework based on reinforcement learning and verifies its significant advantages in resource utilization, service quality and system stability through experiments. The experiment builds a test environment based on the open-source Cassandra cluster and uses YCSB tools to simulate three typical load scenarios: mixed read-write, burst traffic and long-term fluctuation. Compared with traditional consistent hash, weighted polling and threshold triggering strategies, the proposed method is comprehensively evaluated. Performance improvement effect.

(1) In the read-write mixed load scenario, the reinforcement learning strategy successfully reduced the incidence of hot sharding in a 10-node cluster from 23. 7% in the traditional method to 13. 8% by dynamically sensing the sharded data distribution and query mode, a decrease of 42%. At the same time, the proportion of cross-sharded queries was reduced by 31%, and the average query delay was optimized from 152ms to 99ms.

(2) Agents can effectively learn the correlation law between data locality and load distribution and reduce cross-node access overhead through real-time fine-tuning of sharding boundaries. For burst traffic scenarios, the traditional threshold trigger mechanism causes 28% additional network load due to frequent migration, while the reinforcement learning framework reduces the data migration volume to 72% of the benchmark solution by predicting the load trend and the migration cost trade-off, and the migration operation is concentrated during low load periods, and the system jitter time is shortened by 64%. In the 24-hour unsteady load stress test, the throughput standard deviation of the reinforcement learning strategy is 112ops, which is 61% lower than the 289ops of the traditional method, showing stronger robustness.

(3) Experiments further verify the effectiveness of state representation and reward function design. By introducing the timing characteristics of node CPU utilization, memory footprint, network queue depth and fragment query density, the agent can capture the key load inflexion point in 90% of the decision cycle, avoiding the lag of traditional monitoring indicators. The compound reward function combines throughput weight (0. 6), delay penalty (0. 3) and migration cost coefficient (0. 1), which enables the system to reduce the variance of node resource utilization from 0. 38 to 0. 15 and improve the balance degree by 60% under the condition of ensuring service quality. In addition, the application of a double delay depth deterministic strategy gradient (TD3) algorithm solves the balance problem between action space continuity and exploration noise. After 500,000 training steps, the strategy convergence speed is 2. 3 times faster than that of traditional DQN, and the long-term revenue has increased by 19%.

The dynamic optimization framework driven by reinforcement learning can break through the rigid constraints of static policies and realize the joint optimization of sharding and load balancing in complex and changeable load environments. Experimental data confirm that this method has significant advantages in reducing operation and maintenance costs and improving system flexibility, providing theoretical support and practical examples for the intelligent evolution of distributed databases. Future research will further explore multi-objective optimization and heterogeneous hardware adaptation issues and promote the large-scale application of autonomous database systems in industrial scenarios.

References

[1] M. A. Abdel-Fattah, W. Mohamed, and S. Abd elgaber, “A Comprehensive Spark-Based Layer f or Converting Relational Databases to NoSQL,” Big Data and Cognitive Computing, vol. 6, no. 3, 2022. https://doi.org/10.3390/bdcc6030071

[2] U. Al-Qarni Anwar Ridwan, T. C. Chuah, and Y. L. Lee, “EVOLUTIONARY NETWORK SLI CE ASSOCIATION ALGORITHM FOR LOAD BALANCING IN HETEROGENEOUS OPEN R ADIO ACCESS NETWORKS, ” Journal of Eng ineering Science and Technology, vol. 19, no. 1 , 2024.

[3] D. G. Chandra, "BASE analysis of NoSQL data base," Future Generation Computer Systems-the International Journal of Escience, vol. 52, no., pp. 13-21, 2015. https://doi.org/10.1016/j.future.2 015.05.003

[4] K. Hejja, S. Berri, and H. Labiod, “Network sli cing with load-balancing for task offloading in vehicular edge computing,” Vehicular Communi cations, vol. 34, 2022. https://doi.org/10.1016/j.v ehcom.2021.100419

[5] B. Lamrani, Y. Elmrabet, I. Mathew, N. Bekkio ui, P. Etim, A. Chahboun, A. Draoui, and M. C. Ndukwu, “Energy, economic analysis and m athematical modelling of mixed-mode solar dryi ng of potato slices with thermal storage loaded V-groove collector: Application to Maghreb regi on,” Renewable Energy, vol. 200, 2022. https:// doi.org/10.1016/j.renene.2022.09.119

[6] H. Li, Q. Sheng, Q. Xie, Y. Xie, “Inversion of Lateral Impact Load Migration History and Tr ajectory Location Method for Deck Structures B ased on the Time-Slicing Method,” Shock and Vibration, vol. 2024, 2024. https://doi.org/10.115 5/2024/6663995

[7] W. Lu, X. Chen, “Short-term load forecasting f or power systems with high-penetration renewab les based on multivariate data slicing transforme r neural network,” Frontiers in Energy Research , vol. 12, 2024. https://doi.org/10.3389/fenrg.202 4.1355222

[8] T. Misugi, H. Miura, K. Hirata, T. Tachibana, “Design of Multiple Routing Configurations Co nsidering Load Distribution for Network Slicing, ” Apsipa Transactions on Signal and Informati on Processing, vol. 12, no. 2, 2023. https://doi.o rg/10.1561/116.00000148

[9] T. Niaz, S. Shabbir, T. Noor, M. Imran, “Active Composite Packaging Reinforced with Nisin-Loaded Nano-Vesicles for Extended Shelf Life of Chicken Breast Filets and Cheese Slices,” Food and Bioprocess Technology, vol. 15, no. 6, 2022. https://doi.org/10.1007/s11947-022-02815-2

[10] G. C. Deka, "NoSQL Polyglot Persistence," Deep Dive into Nosql Databases: The Use Cases and Applications, Advances in Computers P. Raj and G. C. Deka, eds., pp. 357-390, 2018. https://doi.org/10.1016/bs.adcom.2017.08.003

[11] D. Chen, Z. Chen, Z. He, J. Gao, and Z. Su, “Learning heuristics for weighted CSPs through deep reinforcement learning,” Applied Intelligence, vol. 53, no. 8, pp. 8844-8863, 2023. https://doi.org/10.1007/s10489-022-03992-5

[12] A. Aggoune, and M. S. Namoune, “P3 Process for Object-Relational Data Migration to NoSQL Document-Oriented Datastore,” International Journal of Software Science and Computational Intelligence-ljssci, vol. 14, no. 1, 2022. https://doi.org/10.4018/ijssci.309994

[13] H. Akid, G. Frey, M. Ben Ayed, and N. Lachiche, “Performance of NoSQL Graph Implementations of Star vs. Snowflake Schemas,” leee Access, vol. 10, pp. 48603-48614, 2022. https://doi.org/10.1109/access.2022.3171256

[14] C. Araujo, M. Oliveira Jr, B. Nogueira, P. Maciel, and E. Tavares, “Performability evaluation of NoSQL-based storage systems,” Journal of Systems and Software, vol. 208, 2024. https://doi.org/10.1016/j.jss.2023.111885

[15] S. Bouaziz, S. Boukettaya, A. Nabli, and F. Gargouri, “A formal algebra for document-oriented NoSQL data warehouses: formalisation and evaluation,”Cluster Computing-the Journal of Networks Software Tools and Applications, vol. 28, no. 3, 2025. https://doi.org/10.1007/s10586-024-04776-x

[16] I. Carvalho, F. Sa, and J. Bernardino, “Performance Evaluation of NoSQL Document Databases :Couchbase, CouchDB, and MongoDB,” Algorithms, vol. 16, no. 2, 2023. https://doi.org/10.3390/a16020078

[17] S. H. Wu, and T. A. Mueller, “A user-friendly NoSQL framework for managing agricultural field trial data,” Scientific Reports, vol. 14, no. 1, 2024. https://doi.org/10.1038/s41598-024-81609-2

[18] S. Ferreira, J. Mendonca, and E. Andrade, “Experimental Performance Analysis of Data Consistency Levels in NoSQL Databases,” Software-Practice& Experience, vol. 55, no. 6, 2025. https://doi.org/10.1002/spe.3412

[19] L. H. Zambom Santana, and R. d. S. Mello, “Persistence of RDF Data into NoSQL: A Survey and a Reference Architecture,” leee Transactions on Knowledge and Data Engineering, vol. 34, no. 3, pp. 1370-1389, 2022. https://doi.org/10.1109/tkde.2020.2994521

[20] A. K. Samanta, and N. Chaki, “An enumerated analysis of NoSQL data models using statistical tools,” Innovations in Systems and Software Engineering, vol. 19, no. 1, pp. 5-14, 2023. https://doi.org/10.1007/s11334-022-00517-8

[21] N. Gupta, and R. Agrawal, "NoSQL Security," Deep Dive into Nosql Databases: The Use Cases and Applications, Advances in Computers P. Raj and G. C. Deka, eds., pp. 101-132, 2018. https://doi.org/10.1016/bs.adcom.2018.01.003

[22] X. Li, H. Xu, J. Zhang, and H. -h. Chang, “Deep Reinforcement Learning for Adaptive Learning Systems,” Journal of Educational and Behavioral Statistics, vol. 48, no. 2, pp. 220-243, 2023. https://doi.org/10.3102/10769986221129847

[23] Z. Li, X. Wang, L Pan, L Zhu, Z. Wang, J. Feng, C. Deng, and L Huang, “Network Topology Optimization via Deep Reinforcement Learning,” leee Transactions on Communications, vol. 71, no. 5, pp. 2847-2859, 2023. https://doi.org/10.1109/tcomm.2023.3244239

[24] Y. Liu, X. Wu, Y. Bo, J. Wang, and L Ma, “A Transfer Learning Framework for Deep Multi-Agent Reinforcement Learning,” leee-Caa Journal of Automatica Sinica, vol. 11, no. 11, pp. 2346-2348, 2024. https://doi.org/10.1109/jas.2023.124173

[25] Y. Matsuo, Y. LeCun, M. Sahani, D. Precup, D. Silver, M. Sugiyama, E. Uchibe, and J. Morimoto, “Deep learning, reinforcement learning, and world models,” Neural Networks, vol. 152, pp. 267-275, 2022. https://doi.org/10.1016/j.neunet.2022.03.037

[26] S. D. Kuznetsov, and A. V. Poskonin, "NoSQL data management systems," Programming and Computer Software, vol. 40, no. 6, pp. 323-332, 2014. https://doi.org/10.1134/s0361768814060152

[27] Q. Sun, X. Wei, and X. Yang, “GraphSAGE with deep reinforcement learning for financial portfolio,” Expert Systems with Applications, vol. 238, 2024. https://doi.org/10.1016/j.eswa.2023.122027

[28] W. Wang, B. Li, X. Luo, and X. Wang, “Deep Reinforcement Learning for Sequential Targeting,” Management Science, vol. 69, no. 9, pp. 5439-5460, 2023. https://doi.org/10.1287/mnsc.2022.4621

[29] X. Wang, S. Wang, S. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao, “Deep Reinforcement Learning: A Survey,” leee Transactions on Neural Networks and Learning Systems, vol. 35, no. 4, pp. 5064-5078, 2024. https://doi.org/10.1109/tnnls.2022.3207346

[30] K. Wu, H. Wang, M. A. Esfahani, and S. Yuan, “Learn to Navigate Autonomously Through Deep Reinforcement Learning,” leee Transactions on Industrial Electronics, vol. 69, no. 5, pp. 5342-5352, 2022. https://doi.org/10.1109/tie.2021.3078353